วิธีกำจัดโปรแกรมประเภท TSR

ขจร โรจนเมธินทร์

ผมคิดว่าผู้อ่านทุกท่านคงเคยใช้โปรแกรมไซด์คิกมาบ้าง ซึ่งเป็นโปรแกรมประเภท TSR (Terminate and stay resident) คือ หลังจากที่โปรแกรมจบการทำงานแล้ว โปรแกรมจะฝังตัวบนหน่วยความจำ คราวนี้ถ้าเราอยากกำจัดโปรแกรมประเภทนี้ออกจากหน่วยความจำ ก็จะสามารถทำได้วิธีเดียวคือบูตเครื่องใหม่ ซึ่งวิธีการนี้ผมเห็นว่าไม่ค่อยสะดวกเท่าไหร่ในแง่การทำงาน

ย้อนหลังกลับไปสมัยที่ดอสเวอร์ชัน 1.0 เริ่มออกมาใช้งาน ดอสรุ่นนี้ยังไม่สามารถรองรับการสร้างโปรแกรมประเภท TSR และเมื่อดอสเวอร์ชัน 1.1 ออกวางตลาด ได้มีการเพิ่มฟังก์ชันสำหรับที่จะสร้างโปรแกรมประเภท TSR จนถึงปัจจุบัน (ดอสเวอร์ชัน 3.3) มีบริการเกี่ยวกับ TSR อยู่ 2 แบบ คือ อินเตอร์รัพต์ 27h และ อินเตอร์รัพต์ 21h หมายเลข 31h แต่อย่างไรก็ตามดอสไม่ได้สร้างอินเตอร์รัพต์ที่ทำงานย้อนกลับกับอินเตอร์รัพต์ทั้งสองแบบมาให้ใช้เลย คือ ไม่มีอินเตอร์รัพต์ที่จะใช้ยกเลิกการทำงานของโปรแกรมและนำเอาเนื้อที่ของหน่วยความจำบริเวณที่เคยถูกโปรแกรม TSR ใช้นั้นกลับมาใช้งานใหม่

ในบทความนี้ผมจะเสนอยูทิลิตี้ 2 โปรแกรมคือ INSTALL และ REMOVE ซึ่งจะทำงานคู่กันโดยคุณจะต้องรัน INSTALL ก่อนที่คุณจะโหลดโปรแกรมพวก TSR เสมอ จากนั้นถ้าต้องการกำจัดโปรแกรมนั้นออกก็ทำได้โดยรัน REMOVE สำหรับการทำงานของ REMOVE จะเป็นแบบ LIFO คือ โปรแกรมที่ถูกโหลดลงบนหน่วยความจำหลังสุดจะถูกกำจัดออกก่อน

บริการของดอสที่เกี่ยวกับการจัดการหน่วยความจำ

ดอสจะมองเห็นหน่วยความจำทั้งหมดโดยแบ่งออกเป็นบล็อก สำหรับขนาดของแต่ละบล็อกมีได้ตั้งแต่ 16 ไบต์ ถึง 1 เมกะไบต์ นอกจากนั้นดอสยังต้องใช้หน่วยความจำอีกส่วนหนึ่งเรียกว่า Memory control block (MCB) สำหรับเก็บความยาวของบล็อกและหมายเลขของโปรแกรมที่ใช้หน่วยความจำบล็อกนั้นอยู่ส่วนบริการที่ดอสมีให้ใช้โดยเรียกผ่านอินเตอร์รัพต์ 21h มีดังนี้คือ

1. การจองหน่วยความจำ (Allocate memory)

ก่อนเรียกใช้

AH = 48h

BX = จำนวนพารากราฟที่ขอจองใช้ (1 พารากราฟ = 16 ไบต์)

หลังเรียกใช้

AX : 0 = เซกเมนต์แอดเดรสของบล็อกที่จองได้

AX = Error code ถ้า Carry flag ถูกเซต

BX = ขนาดของบล็อกใหญ่ที่สุดที่จองได้ กรณีที่มีการผิดพลาดเกิดขึ้น

2. การคืนเนื้อที่บนหน่วยความจำที่จองไว้ (Free allocated memory)

ก่อนเรียกใช้

AH = 49h

ES = เซกเมนต์แอดเดรสของบล็อกที่จะคืน

หลังเรียกใช้

AX = Error code ถ้า Carry flag ถูกเซต

3. การปรับขนาดของบล็อกที่จองไว้ (Modify allocated memory block)

ก่อนเรียกใช้

AH = 4Ah

BX = ขนาดของบล็อกใหม่ที่จะปรับเป็นจำนวนของพารากราฟ

หลังเรียกใช้

BX = ขนาดของบล็อกใหม่ที่จะปรับเป็นจำนวนของพารากราฟ

หลังเรียกใช้

AX = Error code ถ้า Carry flag ถูกเซต

BX = ขนาดของหน่วยความจำที่มากที่สุดจะขยายได้ กรณีที่มีการผิดพลาดเกิดขึ้น

เบื้องหลังการเอ็กซีคิวต์โปรแกรมของดอส

ถ้าคุณต้องการโหลดโปรแกรมเพื่อทำงานขณะอยู่ที่ดอสพรอมต์นั้น ดอสจะต้องทำการจองเนื้อที่บนหน่วยความจำมา 2 ส่วน คือ

บริเวณแรกคือ Environment Block (EB) ของโปรแกรมซึ่งจะเก็บรายละเอียดของเอนไวรอนเมนต์ของระบบขณะก่อนจะทำการเอ็กซีคิวต์โปรแกรม โดยจะเก็บในรูปแบบ "Environment-variable = Variable-value" ได้แก่ Path name ที่ถูกกำหนดด้วยคำสั่ง PATH เช่น PATH = \ BINARY เป็นต้น Path ที่จะเข้าถึง Command processor เช่น COMSPEC = A: \ COMMAND.COM เป็นต้น สำหรับดอสเวอร์ชัน 3.0 ขึ้นไป ชื่อของโปรแกรมที่เอ็กซีคิวต์จะถูกเก็บที่บริเวณท้ายสุดของเอนไวรอนเมนต์บล็อกด้วย ขนาดของหน่วยความจำส่วนนี้ใช้ไม่แน่นอนขึ้นกับว่ามีตัวแปรมากแค่ไหน แต่ในดอสเวอร์ชัน 3.2 ได้กำหนดไว้ไม่เกิน 32 กิโลไบต์

บริเวณที่สองนี้เป็นที่อยู่ของโปรแกรมที่จะเอ็กซีคิวต์ โดยมี Program Segment Prefix (PSP) เริ่มที่ออฟเซต 0 ขนาด 256 ไบต์ แล้วตามด้วยโค้ดและดาต้าของโปรแกรม กรณีที่เป็นไฟล์ชนิด .COM ดอสจะจองเนื้อที่บนหน่วยความจำไว้ 64 กิโลไบต์เสมอ แต่สำหรับไฟล์ชนิด .EXE ดอสจะจองเนื้อที่ตามขนาดที่โปรแกรมใช้จริงตามที่กำหนดใน File Header สิ่งสำคัญใน PSP คือที่ออฟเซต 2Ch ของ PSP จะเก็บค่าเซกเมนต์แอดเดรสของเอนไวรอนเมนต์บล็อกไว้ด้วย

สำหรับที่อยู่ของหน่วยความจำทั้งสองนี้ ถ้าดอสสามารถจองหน่วยความจำได้ในบริเวณที่มีขนาดใหญ่พอและอยู่ติดต่อกันตลอด ดอสจะกำหนดให้เอนไวรอนเมนต์บล็อกอยู่นำหน้าส่วนของ PSP บล็อกทันที แต่ถ้ามีเนื้อที่บนหน่วยความจำที่เป็นบริเวณเล็ก (Hole) ที่มีขนาดใหญ่พอ ดอสก็จะแยกเอาเอนไวรอนเมนต์บล็อกไปไว้ที่บริเวณนั้นแทน

หลังจากที่ดอสจองเนื้อที่ทั้งสองได้แล้ว ดอสก็จะเรียกใช้ฟังก์ชัน EXEC เพื่อมาเอ็กซีคิวต์โปรแกรมต่อไป

โครงสร้างของ MCB

ดอสสร้าง MCB ขึ้นมาเพื่อใช้ควบคุมการใช้เนื้อที่ของหน่วยความจำทั้งหมดโดย MCB แต่ละอันใช้เนื้อที่ 16 ไบต์ สำหรับส่วนประกอบเป็นตามรูปที่ 1 คือ

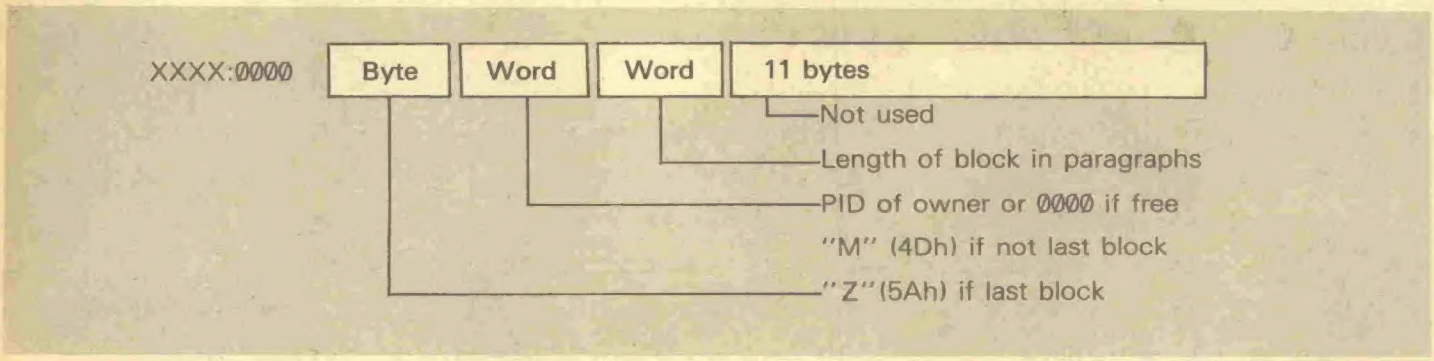
ไบต์ 0 จะมีค่า 4Dh ("M") หรือ 5Ah ("Z") ถ้าเป็น 4Dh แสดงว่า MCB นั้นไม่ได้เป็น MCB สุดท้าย แต่ถ้าเป็น 5Ah จะหมายถึงว่า MCB นั้นเป็น MCB ที่ควบคุมหน่วยความจำ Block สุดท้าย

ไบต์ 1-2 จะเก็บเลขที่ของ Process (Process ID. PID) ซึ่งใช้หน่วยความจำส่วนที่ MCB นั้นควบคุมอยู่ โดยทั่วไป PID นี้ก็คือ PSP เซกเมนต์แอดเดรสของโปรแกรมที่ใช้หน่วยความจำบล็อกที่ MCB นั้นควบคุมอยู่

ไบต์ 3-4 จะเก็บจำนวนพารากราฟของหน่วยความจำที่ MCB ควบคุมอยู่

ไบต์ 5-15 ไม่ได้ใช้

เพราะฉะนั้นถ้าเรารู้ว่า MCB อันแรกอยู่ที่ไหน เราก็จะสามารถหา MCB ถัดไปได้โดยนำค่าในไบต์ที่ 3 มาบวกกับค่าเซกเมนต์แอดเดรสของ MCB นั้นแล้วบวกหนึ่ง เราก็จะได้ค่าเซกเมนต์แอดเดรสของ MCB ถัดมา ดังนั้นเราก็จะสามารถไล่ตรวจสอบหน่วยความจำทั้งหมดที่มีได้



รูปที่ 1 โครงสร้าง MCB (16 ไบต์)

การใช้ INSTALL และ REMOVE

สมมติว่าคุณต้องการโหลดโปรแกรมที่เป็น TSR 2 โปรแกรม คือ KBX.COM และ SK.COM คุณจะต้องรัน INSTALL ก่อนเสมอเพื่อเก็บรายละเอียดต่าง ๆ ที่จำเป็นต้องใช้ตอนรัน REMOVE ต่อจากนั้นจึงทำการโหลดโปรแกรมตามปกติ (โปรแกรม INSTALL และ REMOVE ดูรูปที่ 2 และ 3)

รูปแบบคำสั่ง INSTALL [Name Routine]

NameRoutine คือ ชื่อที่จะใช้เรียกโปรแกรมที่คุณจะโหลด ซึ่งไม่จำเป็นต้องเป็นชื่อไฟล์โปรแกรม เช่น โปรแกรมไซด์คิกนั้นไฟล์ชื่อ SK.COM คุณอาจใช้ NameRoutine เป็น "The SideKick Program" ก็ได้

เมื่อคุณต้องการกำจัดโปรแกรมที่ฝังอยู่บนหน่วยความจำ คุณจะต้องรัน REMOVE จากตัวอย่างในรูปที่ 4 เมื่อรัน REMOVE แล้วจะเกิดผลดังรูปที่ 5

REMOVE จะคอยรับคำสั่งจากคุณ ซึ่งถ้าคุณกด ENTER โปรแกรม SK.COM ถูกกำจัดจากหน่วยความจำออกก่อน

รูปที่ 2 โปรแกรมแอสเซมบลี INSTALL

```
;-----------------------------------------------------------------------------
;       INSTALL.COM for the IBM Personal Computer - 1989
;-----------------------------------------------------------------------------
code    segment para public 'code'
        assume cs:code
        org    100h
begin:  jmp    install

tsr_count     dw 0              ;number of TSR's installed
segment_table dw 64 dup (?)     ;wedge program segment
vector_table  db 1024 dup (?)   ;vector table storage area
first_call    db 0              ;INSTALL status flag
old1Ah        label dword       ;storage for the old int 1Ah address
old1Ah_vector dw 2 dup (?)
errmsg        db 'No room for more',13,10,'$'
copyright     db 'The INSTALL Utility, Copyright(C) 1989, Si''99 No:011'
              db 13,10,'$'

;-----------------------------------------------------------------------------
; NEW1AH handles the new function of int 1Ah if AH is 99h.
; Return :      ES = Segment address of Installation Data Table (IDT)
;-----------------------------------------------------------------------------
new1Ah  proc   near
        sti                     ;enable interrupt
        cmp    ah,99h           ;is the function code 99h?
        je     new1Ah1          ;yes, then handle this call
        jmp    old1Ah           ;no, goto the old int 1Ah
new1Ah1: push  cs               ;place current code segment in ES
        pop    es
        dec    bh               ;decrement BH to verify call
        iret                    ;return to calling routine
new1Ah  endp
;-----------------------------------------------------------------------------
; INSTALL saves the data needed for deallocation.
;-----------------------------------------------------------------------------
install proc   near
        assume ds:code
        lea    dx,copyright
        mov    ah,9             ;print copyright
        int    21h
        mov    ah,99h           ;call int 1Ah, function 99h
        xor    bh,bh            ;BH = 0
        int    1Ah
        cmp    bh,0FFh          ;BH = -1?
        je     install1         ;yes, then INSTALL is resident
        mov    first_call,1     ;no, set first_call flag
install1:
        lea    di,tsr_count     ;point DI to installation count
        cmp    word ptr es:[di],32      ;still room for more TSR's?
        jb     install2         ;yes, then continue
        mov    ah,9             ;no, then print error
        lea    dx,errmsg
        int    21h
        mov    ax,4C01h         ;terminate with error AL = 1
        int    21h
install2:
        mov    ax,es:[di]       ;get TSR count in AX
        inc    word ptr es:[di]         ;increment count
        mov    cl,2             ;multiply by 4
        shl    ax,cl
        add    ax,offset segment_table ;complete offset address
        mov    di,ax            ;move to DI
        mov    es:[di],cs       ;save current code segment value
        mov    ax,cs:[2Ch]      ;get environment block segment
        mov    es:[di+2],ax     ;store it
```

รูปที่ 2 (ต่อ)

```
            push    ds              ;svae DS
            xor     ax,ax           ;point DS:SI to int vector table
            mov     ds,ax
            assume  ds:nothing
            xor     si,si
            push    cs
            pop     es
            lea     di,vector_table
            mov     cx,512          ;move 512 words
            cld                     ;clear DF
            cli                     ;suspend interrupt during move
            rep     movsw           ;copy vector table
            sti                     ;restore interrupt
            pop     ds
            assume  ds:code

            cmp     first_call,0    ;is the first call?
            je      install3        ;no, then skip ahead
            mov     ax,351Ah        ;get address of int 1Ah
            int     21h
            mov     old1Ah_vector,bx        ;save
            mov     old1Ah_vector[2],es
            mov     ah,25h          ;set the new int 1Ah to NEW1AH
            lea     dx,new1Ah
            int     21h
install13:
            lea     dx,install      ;point DX to end
            int     27h             ;TSR
install     endp

code        ends
            end     begin
```

รูปที่ 3 โปรแกรมแอสเซมบลี REMOVE

```
;-----------------------------------------------------------------------------
;       REMOVE.COM for The IBM Personal Computer - 1989
;-----------------------------------------------------------------------------
tsr_count       equ 0103h       ;pointer to installation count
segment_table   equ 0105h       ;pointer to wedge segment value
vector_table    equ 0185h       ;pointer ro vector table

code        segment para public 'code'
            assume cs:code,ds:code
            org     100h
begin:      jmp     remove

copyright       db 'The REMOVE Utility, Copyright(C) 1989, Si''99 No:011'
                db 13,10,'$'
errmsg1         db 13,10,'None installed',13,10,'$'
errmsg2         db 13,10,'Deinstallation failed',13,10,'$'
msg1            db 13,10,'Number of installation: $'
msg2            db 13,10,'Press ENTER to remove, ESC to abort',13,10,'$'
msg3            db 13,10,'Deinstalltion completed',13,10,'$'
data_segment    dw ?            ;Installation Data Table segment
last_segment    dw ?            ;temporary segment storage
mcb_start       dw ?            ;segment address of first MCB
count           dw ?            ;number of TSR's installed
;-----------------------------------------------------------------------------
; REMOVE procedure is the main body of the program.
;-----------------------------------------------------------------------------
remove      proc    near

            xor     al,al           ;blank screen
            xor     cx,cx           ;upper left corner = (0,0)
            mov     dx,184Fh        ;lower right corner = (79,24)
            mov     bh,7            ;normal attribute
            mov     ah,6            ;scroll up function
            int     10h             ;clear screen
            xor     dx,dx           ;set (x,y) = (0,0)
            xor     bh,bh           ;page 0
            mov     ah,2            ;set cursor position
            int     10h             ;go to (x,y)
            lea     dx,copyright
            call    write_string
            mov     ah,99h          ;call int 1Ah, function 99h
            xor     bh,bh           ;BH = 0
            int     1Ah
            cmp     bh,0FFh         ;BH = -1?
            je      remove1         ;yes, there's at least one TSR installed
            lea     dx,errmsg1      ;no, there's nothing to remove
            call    write_string    ;print error message
            mov     al,1            ;error set up
```

รูปที่ 3 (ต่อ)

```
exit:     mov    ah,4Ch          ;terminate
          int    21h

remove1:  lea    dx,msg1         ;print number of installation
          call   write_string
          mov    ax,es:[tsr_count]       ;retrieve installation count
          mov    count,ax        ;save it
          call   write_number    ;print the installation count
          call   cr_lf           ;next 2 lines
          call   cr_lf
          call   write_tsr_list          ;print list of TSR's installation
          lea    dx,msg2         ;verify to proceed
          call   write_string
remove2:  mov    ah,0            ;get user command
          int    16h
          cmp    al,13           ;ENTER pressed?
          je     remove3         ;yes, proceed
          cmp    al,27           ;ESC pressed?
          jne    remove2         ;no, try again
          xor    al,al           ;no error exit
          jmp    exit            ;terminate
remove3:  push   ds              ;save DS
          mov    ds,last_segment         ;point DS to wedge PSP segment
          assume ds:nothing
          mov    si,vector_table         ;point SI to stored vectors
          mov    data_segment,es         ;store IDT segment
          xor    ax,ax           ;point ES:DI to main vector table
          mov    es,ax
          xor    di,di
          mov    cx,512          ;512 words to move
          cli                    ;disable interrupt
          rep    movsw           ;restore int vector table
          sti                    ;enable interrupt
find:     mov    es,ax           ;point ES to next segment
          inc    ax              ;point AX to the one after it
          cmp    byte ptr es:[0],4Dh     ;is MCB signature there("M")?
          jne    find            ;no, then loop
          cmp    ax,es:[1]       ;does next segment own this one?
          jne    find            ;no, then loop
          mov    mcb_start,es    ;yes, store it
          mov    ds,data_segment         ;set DS to IDT segment
          mov    si,count        ;calculate pointer into IDT
          dec    si              ;adjust to index
          mov    cl,2
          shl    si,cl           ;SI = SI*4
          add    si,segment_table
          mov    bx,[si]         ;PSP segment of last wedge
remove4:  dec    bx              ;address memory control block
          mov    es,bx
          add    bx,es:[3]       ;calculate address of next MCB
          inc    bx              ;now BX point to next MCB
          mov    es,bx           ;store in ES
          inc    bx              ;advance to block segment
          cmp    bx,es:[1]       ;is this a PSP block?
          jne    remove4         ;no, then advance to next block
          mov    ax,cs           ;get current PSP segment
          cmp    bx,ax           ;have we reached our own PSP?
          je     remove7         ;yes, then exit loop
          mov    ax,mcb_start    ;retrieve address of first MCB
remove5:  mov    es,ax           ;point ES to MCB
remove6:  add    ax,es:[3]       ;calculate address of next MCB
          inc    ax
          mov    es,ax           ;store in ES
          cmp    byte ptr es:[0],5Ah     ;is the last block of chain("Z")?
          je     remove4         ;yes, then exit loop
          cmp    bx,es:[1]       ;does the ownership word match?
          jne    remove6         ;no, then try next block
          mov    dx,ax           ;save segment value in DX
          inc    ax              ;advance AX to PSP block
          mov    es,ax           ;tranfer to ES
          mov    ah,49h          ;deallocate block
          int    21h
          jc     mem_error       ;have some error
          mov    ax,dx           ;restore to AX
          jmp    remove5         ;loop again
remove7:  mov    es,[si+2]       ;point ES to wedge env.segment
          mov    ah,49h
          int    21h
          jc     mem_error       ;have some error
          mov    es,[si]         ;point ES to wedge PSP segment
          mov    ah,49h
          int    21h
          jnc    remove8         ;no any error
mem_error:
          pop    ds              ;set DS to code segment
          assume ds:code
          lea    dx,errmsg2      ;print error message
          call   write_string
          mov    al,1            ;error set up
          jmp    exit
```

→

รูปที่ 3 (ต่อ)

```
remove8: pop    ds
         mov    es,data_segment          ;recover IDT segment
         dec    word ptr es:[tsr_count] ;decrement installation count
         lea    dx,msg3                  ;print message and exit
         call   write_string
         xor    al,al                    ;no error exit
         jmp    exit
remove   endp
;-------------------------------------------------------------------------
; WRITE_NUMBER writes the number held in AL to the display.
;-------------------------------------------------------------------------
write_number    proc    near
         aam                    ;covert in AL to BCD in AX
         add    ax,3030h        ;binary to ascii
         cmp    ah,30h          ;first digit is zero?
         je     write1          ;yes, then do not write
         push   ax
         mov    dl,ah           ;digit for write
         mov    ah,2
         int    21h
         pop    ax
write1:  mov    dl,al           ;another digit
         mov    ah,2
         int    21h
         ret
write_number    endp
;-------------------------------------------------------------------------
; WRITE_TSR_LIST writes a list of all TSR's present in memory.
; Entry :         ES = Installation Data Table segment
; Output :        LAST_SEGMENT set to the last PSP of wedge
;-------------------------------------------------------------------------
write_tsr_list  proc    near
         cld                    ;clear DF
         mov    cx,count        ;get count of TSR's installed
         mov    di,segment_table        ;point DI to wedge segment values
         push   ds
         assume ds:nothing
tsr_list1:
         mov    ds,es:[di]      ;get wedge segment address in DS
         mov    last_segment,ds         ;store it for later
         mov    si,82h          ;point SI to command line
         cmp    byte ptr [si-2],1       ;more than 1 character entered?
         jbe    tsr_list3       ;no, then no name defined
         push   cx              ;save tsr count
         mov    cl,[si-2]       ;get length of tsr name
         xor    ch,ch
         dec    cx
tsr_list2:
         lodsb                  ;get a character in AL
         mov    dl,al
         mov    ah,2
         int    21h
         loop   tsr_list2
         pop    cx
tsr_list3:
         call   cr_lf           ;next line
         add    di,4            ;poit ES:DI to next segment
         loop   tsr_list1
         pop    ds
         assume ds:code
         ret
write_tsr_list  endp
;-------------------------------------------------------------------------
; CR_LF sends a carriage return and line feed.
;-------------------------------------------------------------------------
cr_lf    proc   near
         push   ax
         push   dx
         mov    dl,13           ;carriage return
         mov    ah,2
         int    21h
         mov    dl,10           ;line feed
         mov    ah,2            ;write it
         int    21h
         pop    dx
         pop    ax
         ret
cr lf    endp
;-------------------------------------------------------------------------
; WRITE_STRING writes a string that point in DX.
;-------------------------------------------------------------------------
write_string    proc    near
         push   ax
         mov    ah,9            ;write string that DS:DX already point to it
         int    21h
         pop    ax
         ret
write_string    endp

code     ends
         end    begin
```

■

```
A>INSTALL KBX                    <--- รัน INSTALL แล้วตามด้วยชื่อโปรแกรมที่จะโหลดต่อ
A>KBX                            <--- โหลดโปรแกรม KBX.COM
A>INSTALL The SideKick Program   <--- รันก่อนโหลดไซด์คิก
A>SK                             <--- โหลดโปรแกรม SK.COM
```

รูปที่ 4 ตัวอย่างการรัน INSTALL

```
Number of installation: 2
KBX
The SideKick Program

Press ENTER to remove, ESC to abort
```

รูปที่ 5 ผลจากการรัน REMOVE

การทำงานของ INSTALL

INSTALL ทำหน้าที่สำคัญ 2 อย่าง คือ

1. ก๊อบปี้ตารางที่เก็บแอดเดรสของอินเตอร์รัพต์เวกเตอร์ทั้งหมดไว้ก่อนที่จะถูกเปลี่ยนแปลงโดยโปรแกรม TSR ที่จะโหลดมาภายหลัง เพื่อจะสามารถแก้ค่าแอดเดรสของอินเตอร์รัพต์ที่ถูกเปลี่ยนไปกลับได้ถูกต้องขณะที่เรากำจัดโปรแกรม TSR นั้นออก

2. เก็บค่าเซกเมนต์แอดเดรสของ PSP บล็อก และเซกเมนต์แอดเดรสของเอนไวรอนเมนต์บล็อกขณะที่เรารัน INSTALL

เมื่อรัน INSTALL โปรแกรมจะตรวจว่า INSTALL ถูกโหลดมาก่อนหรือยังโดยเรียก INT 1Ah ฟังก์ชัน 99h ถ้ายังไม่โหลดก็จะตรวจจำนวนโปรแกรมที่อยู่บนหน่วยความจำว่าถึง 32 โปรแกรมหรือยังเพราะจองเนื้อที่บริเวณ SEGMENT__TABLE ไว้เพียง 64 เวิร์ด จากนั้น INSTALL จะเริ่มเก็บค่า PSP เซกเมนต์แอดเดรส และ EB เซกเมนต์แอดเดรสไว้ที่ SEGMENT__TABLE จากนั้นก็จะเก็บตารางอินเตอร์รัพต์เวกเตอร์ทั้งหมดไว้ที่ VECTOR__TABLE และถ้าเป็นการรัน INSTALL ครั้งแรกก็จะต้องแก้ให้ Int 1Ah ชี้มาที่ NEW1AH แทน จะเห็นว่าในการรัน INSTALL ครั้งต่อมา ค่า PSP,EB เซกเมนต์แอดเดรสจะถูกเก็บที่ SEGMENT__TABLE ของ INSTALL ที่รันครั้งแรกเสมอ เพราะ NEW1AH จะคืนค่าเซกเมนต์แอดเดรสของ SEGMENT__TABLE ของ INSTALL ที่รันครั้งแรกมาทาง ES ส่วน VECTOR__TABLE จะเก็บที่ VECTOR__TABLE ของ INSTALL ที่รันขณะนั้น ในการรัน INSTALL แต่ละครั้งจะใช้เนื้อที่ของหน่วยความจำไป 1616 ไบต์

การทำงานของ REMOVE

REMOVE จะตรวจว่า INSTALL ถูกโหลดก่อนหรือยัง ซึ่งถ้าโหลดแล้ว NEW1AH ก็จะคืนค่าเซกเมนต์แอดเดรสของ SEGMENT__TABLE มาทาง ES ซึ่ง REMOVE ก็จะใช้เซกเมนต์แอดเดรสนี้ไปอ่านค่า PSP เซกเมนต์แอดเดรสของ INSTALL ที่ถูกโหลดหลังสุดจาก SEGMENT__TABLE มาเพื่อจะก๊อบปี้ VECTOR__TABLE ซึ่ง INSTALL เก็บไว้ ไปที่เซกเมนต์แอดเดรสที่ศูนย์ จากนั้น REMOVE จะค้นหาจุดเริ่มต้นของ MCB ของหน่วยความจำทั้งหมด เมื่อพบแล้วก็จะค้นหา MCB ที่ควบคุมบริเวณหน่วยความจำที่โปรแกรมซึ่งเรากำลังจะกำจัดมันใช้อยู่ โดยจะไปอ่านค่า PSP เซกเมนต์แอดเดรสของ INSTALL ที่เราโหลดหลังสุดจาก SEGMENT__TABLE มาใช้เป็นจุดเริ่ม ทั้งนี้ก็เพราะดอสจะโหลดโปรแกรมที่เราจะกำจัดที่บล็อกต่อจากบล็อกที่ใช้โหลด INSTALL ทันที การค้นหาจะถือหลักว่า MCB ที่เป็นจุดเริ่มของโปรแกรมที่จะกำจัดนั้นค่าเซกเมนต์แอดเดรสบวกหนึ่งของมันจะต้องเท่ากับค่า PID ที่ MCB นั้นเก็บอยู่และเมื่อได้ค่าเซกเมนต์แอดเดรสของ MCB ที่ควบคุมหน่วยความจำบริเวณที่โปรแกรมซึ่งเรากำลังจะกำจัดฝังตัวอยู่ เราก็จะทราบค่า PSP เซกเมนต์แอดเดรสของมันทันที (MCB เซกเมนต์แอดเดรสบวกหนึ่ง) จากนั้นก็จะเริ่มค้นหาเอนไวรอนเมนต์บล็อก (ของโปรแกรมที่จะกำจัด) เพื่อทำการกำจัดมันออก แล้วจึงกำจัด PSP บล็อก การค้นหาจะหยุดเมื่อพบ MCB สุดท้าย(ไบต์ 0 เป็น 5Ah) ซึ่งก็คือบล็อกที่ REMOVE ใช้งานอยู่ จากนั้นโปรแกรมก็จะมากำจัดเนื้อที่ ซึ่ง INSTALL ที่เรารันหลังสุดใช้ ได้แก่บริเวณ PSP บล็อกที่ใช้เก็บ VECTOR__TABLE และเอนไวรอนเมนต์บล็อกออกจากหน่วยความจำ

สรุป

INSTALL และ REMOVE จะทำหน้าที่โหลดและยกเลิกโปรแกรมประเภทที่ฝังตัวอยู่บนหน่วยความจำ โดยที่ถ้าคุณอยากจะกำจัดโปรแกรมเหล่านี้ออกก็สามารถทำได้โดยไม่ต้องบูตเครื่องใหม่